



Malware RCE: Debuggers and Decryptor Development

Michael Ligh, Malicious Code Engineer
Greg Sinclair, Rapid-Response Engineer

iDefense Security Intelligence Services

- How to script a debugger for malware RCE
- Almost all malicious code uses some form of obfuscation to protect
 - » Strings
 - » Configurations
 - » Command-and-Control (C&C) protocols/hosts
 - » Stolen Data
- Save time, leverage the code's own functions
 - » All you have to do is find it
- We'll show you some examples
 - » Kraken, Laqma, Silent Banker, DHTMLSpy, Torpig/MBR

The Hiding Game

Why do malicious code authors obfuscate things?

- » To make RCE significantly more difficult
- » To prevent others from reading the stolen data
- » To prevent others from deciphering the C&C protocol
- » To make building IDS/IPS and anti-virus signatures more difficult

The Hiding Game

Why do we de-obfuscate things?

- » To make RCE easier
- » To recover stolen data and credentials
- » To decipher the C&C protocol
- » To build IDS/IPS and anti-virus signatures

Decryptor Development Methods

1. Manual RCE: Decryptor development based on ASM

- » Requires the most intimacy with ASM
- » More code + more complexity = more time consuming

2. Assisted RCE: Pressing F5 (Hex-rays)

- » Requires type fix-up often

3. Scripting the debugger: Let the malware do the work

- » More code + more complexity = more time consuming

The best method depends on the project

- » For example, Wireshark plugins require manual or assisted

Scripting the Debugger

- **Generally speaking, there are three classes of scriptable analysis**
 - » **Active**
 - » You control which functions execute and in which order
 - » Redirect EIP (“New origin here”) and “crawl” through code
 - » Example: Silent Banker encoded strings (max code coverage)
 - » **Passive**
 - » Monitor Trojan with function hooks, API or internal
 - » Let it run through (may need !hiddebug)
 - » Example: Kraken network traffic
 - » **Recon and utility scripting**
 - » Kind of a catch-all, hybrid class
 - » Examples include
 - » Scanning an arbitrary file for shell code
 - » Checking shared DLLs for hooks
 - » Unpacking malware or other “protected” binaries

Requirements

- **Required**
 - » **Copy of the Trojan**
 - » **Debugger**
 - » Immunity Debugger + Python
 - » Olly + OllyScript
 - » IDA + IDC/IDAPython
 - » **Basic RCE knowledge**
 - » Unpacking, breakpoints, stepping, running, registers
- **Optional**
 - » **Disassembler**
 - » **Virtual environment**

Demo Summary

Demo Line-up

- » Silent Banker
 - » Decode binary strings
 - » Resolve hash-based API imports
- » Kraken
 - » Print decrypted network traffic
 - » Generate C&C hostnames
- » Laqma
 - » Snoop on shared memory and window messages (IPC)
- » Torpig/MBR
 - » Extract decrypted data from the kernel driver

[Active] Example: SilentBanker Strings Decoder

- Decoding strings requires some basic RCE work
 - » Locate obfuscated strings
 - » Locate function xrefs to the obfuscated string
 - » Does the function look like a decoder?
 - » Yes - analyze parameters/output (registers and stack)
 - » No - keep looking
 - » The following looks like a decoder

```
mov     eax, offset aTnreuxmr ; "tnreuxmr"  
push   eax  
push   eax  
call   sub_100122E8  
mov     eax, offset aUkbr ; "ukbr"  
push   eax  
push   eax  
call   sub_100122E8  
mov     eax, offset aAdudpmjgy ; "adudpmjgy"
```

```
mov     esi, offset aYekorgwfinadhj ; "YEKORGWFIMADHJBCUTZQPXSLUN"  
lea     edi, [ebp+var_1C]  
rep     movsd  
movsw  
movsb  
push   6  
pop     ecx  
mov     esi, offset aSdhvkltrjizgn ; "sdhvkltrjizgnaqfybmueoxwc"  
lea     edi, [ebp+var_38]
```

[Active] Example: SilentBanker Strings Decoder

```
decoder_start = 0x100122E8
ref = imm.getXrefFrom(decoder_start)
# for each function that calls the decoder function
for info in ref:
    xref = info[0]
    mov_eax = 0
    addr = 0
    # disassemble backwards until finding MOV EAX, offset aStringName
    for i in range (1,4):
        op = imm.disasmBackward(xref, i)
        instr = op.getDisasm()
        if "MOV EAX" in instr:
            # get address of encoded string in memory (.text section)
            addr = op.operand[1][3]
            break
    if addr != 0:
        before = imm.readString(addr)
        # prepare the stack for the function call
        imm.setReg('EIP', xref)
        # the input and output buffers can be the same
        imm.writeLong(imm.getRegs()['ESP'], addr)
        imm.writeLong(imm.getRegs()['ESP']+4, addr)
        imm.stepOver()
        # now read the result
        after = imm.readString(addr)
        table.add('', ['0x%x' % addr, '%s' % before, '%s' % after])
```

[Active] Example: SilentBanker Strings Decoder

DEMO

Address	Encoded	Decoded
0x1000145c	QUTRN*UP	WORTH_OF
0x10001cc0	Srlj	Memo
0x10001424	Kteersgk*Mklpjm	Currency_Symbol
0x100015d0	PXLR*TLRB	FIAT_RATE
0x100019b8	FLJBB*HLSB	PAYEE_NAME
0x10001d18	FLJSBHR*LSUOHR	PAYMENT_AMOUNT
0x10001714	FLJSBHR*OHXRM	PAYMENT_UNITS
0x10001c08	FLJSBHR*SBRLA*XU	PAYMENT_METAL_ID
0x10001854	FLJBT*LKKUOHR	PAYER_ACCOUNT
0x10001b68	OMV*FBT*UOHKB	USD_PER_OUNCE
0x100017c0	dtujR2	autoT1
0x10001920	dtujR1	autoT2
0x10001c1c	.ELguxjs)Kjsvkel	&BAction=Confirm
0x100018bc	mjcx:	login:
0x10001a74	bnz:	psw:
0x10001ad0	OMBT	USER
0x1000164c	FLMM	PASS
0x10001dfc	bdenre	parser
0x10001cb4	<455UC55>	<!--OK-->
0x10001c64	MrUrpToFexfxmror	SeDebugPrivilege

[Active] Example: Silent Banker API Resolution

- **The effect of run-time dynamic linking**
 - » LoadLibrary/GetProcAddress instead of IAT
 - » API calls look like “call dword ptr [eax+20h]”
 - » Makes static analysis a PITA
- **SilentBanker uses hash-based RTDL**
 - » Frequently used in shell code
 - » Binary contains 32-bit hash instead of function name
 - » It walks a loaded library's exports
 - » if hash(getNextExportName()) == 0x19272710
 - » Makes static analysis an even bigger PITA

[Active] Example: Silent Banker API Resolution

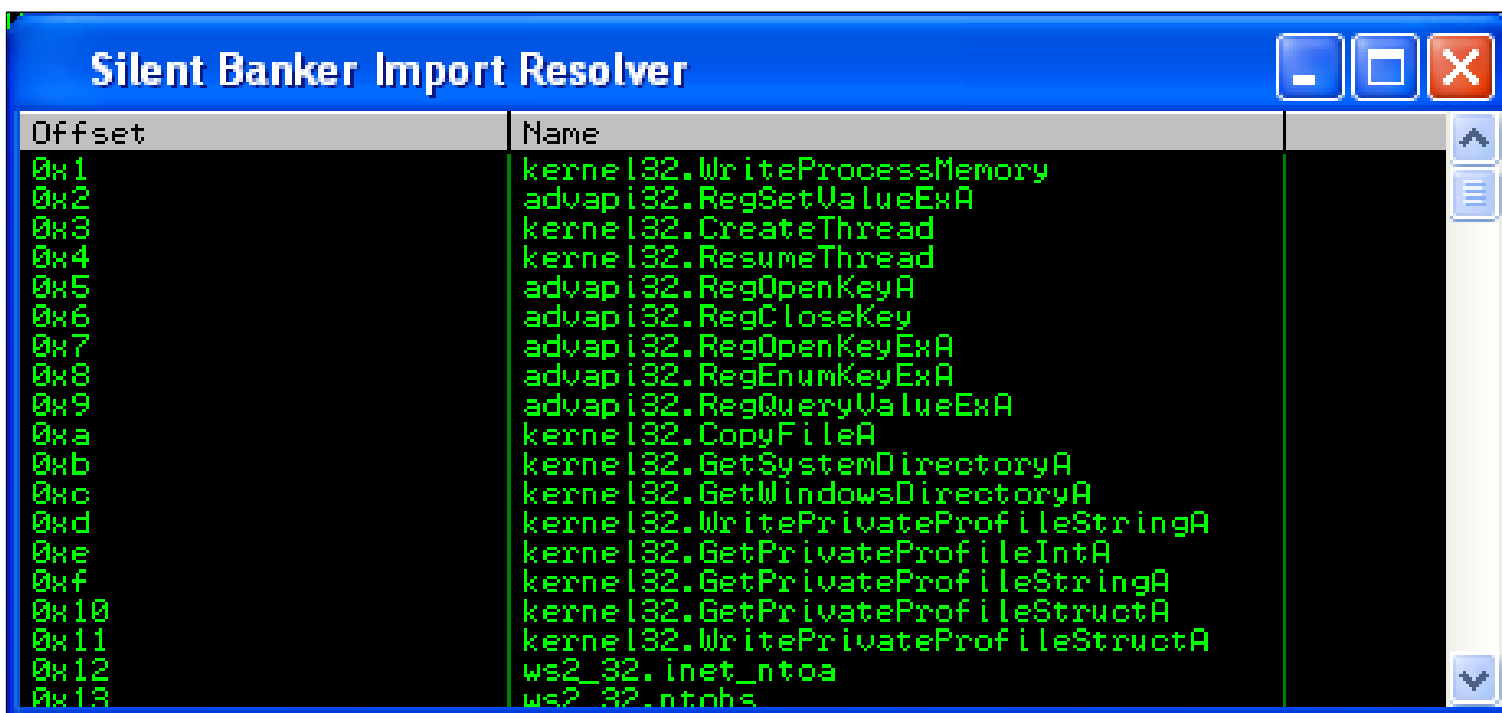
Quick solution to the problem

- » Find the base address of call table
- » Run the Trojan until it fills in the call table
- » Loop through the call table with reverse lookups
- » Add a structure to the IDB and rename if desired
 - » call dword ptr [eax+20h]
 - » call dword ptr [eax+ExitProcess]

```
for i in range (1,100):
    addr = imm.readLong( fxtbl + (i*4) )
    if addr:
        func = imm.getFunction(addr)
        name = func.getName()
        table.add('', ["0x%x" % i, "%s" % name])
```

[Active] Example: Silent Banker API Resolution

DEMO



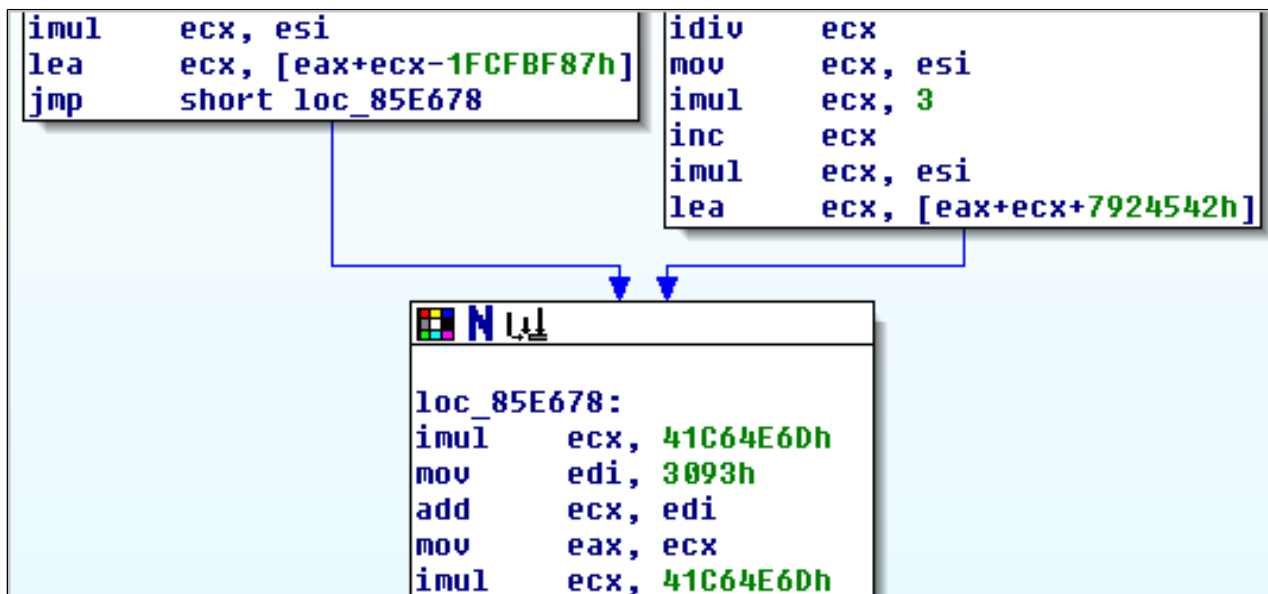
The screenshot shows a window titled "Silent Banker Import Resolver" with a blue title bar and standard Windows window controls. The window contains a table with two columns: "Offset" and "Name". The table lists 14 entries, each with a hexadecimal offset and a corresponding API name. The text is displayed in a green monospaced font on a black background.

Offset	Name
0x1	kernel32.WriteProcessMemory
0x2	advapi32.RegSetValueExA
0x3	kernel32.CreateThread
0x4	kernel32.ResumeThread
0x5	advapi32.RegOpenKeyA
0x6	advapi32.RegCloseKey
0x7	advapi32.RegOpenKeyExA
0x8	advapi32.RegEnumKeyExA
0x9	advapi32.RegQueryValueExA
0xa	kernel32.CopyFileA
0xb	kernel32.GetSystemDirectoryA
0xc	kernel32.GetWindowsDirectoryA
0xd	kernel32.WritePrivateProfileStringA
0xe	kernel32.GetPrivateProfileIntA
0xf	kernel32.GetPrivateProfileStringA
0x10	kernel32.GetPrivateProfileStructA
0x11	kernel32.WritePrivateProfileStructA
0x12	ws2_32.inet_ntoa
0x13	ws2_32.ntohs

[Active] Example: Kraken C&C Hostname Generation

Spam bot uses UDP 447 and HTTP POST for C&C

- » Locates C&C servers based on hostname algorithm
- » It is easy to find the algorithm-containing function
- » Analyze the function parameters
 - » for(i=0; i<1000; i++) { getDomain(&ptr, i); }



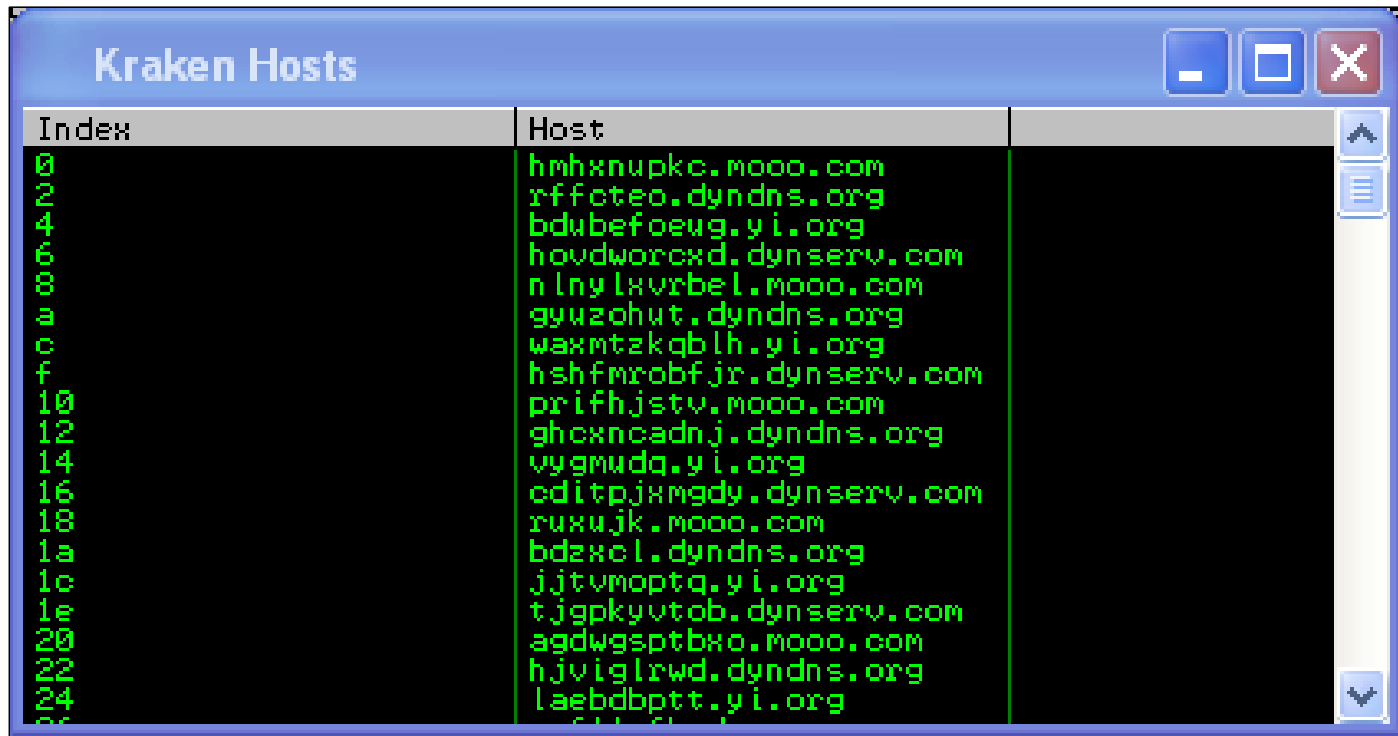
[Active] Example: Kraken C&C Hostname Generation

Make our own loop around the algorithm function

- » Direct EIP to start of function
- » Set breakpoint at end of function
- » Modify stack parameters on each iteration (relative to EBP)
- » Run until the end breakpoint is hit
- » Read the generated hostname string

```
for idx in range(0,2000):  
    imm.setReg("EIP", start)  
    imm.Run() # past function prologue  
    regs = imm.getRegs()  
    imm.writeLong(regs['EBP']+8, memvar)  
    imm.writeLong(regs['EBP']+12, idx)  
    imm.Run() # to end of function  
    host = imm.readString(imm.readLong(memvar))  
    table.add('', ['%x' % idx, '%s' % host])
```


DEMO



The screenshot shows a window titled "Kraken Hosts" with a table of hostnames. The table has two columns: "Index" and "Host". The hostnames are listed in a green monospace font on a black background. The window includes standard Windows-style window controls (minimize, maximize, close) and a vertical scrollbar on the right side.

Index	Host
0	hmhxnpkc.mooo.com
2	rfoteo.dyndns.org
4	bdubefoeg.yi.org
6	hovdworoxd.dynserv.com
8	nlnylxvrbel.mooo.com
a	gyuzohut.dyndns.org
c	waxmtakqblh.yi.org
f	hshfmrobjfr.dynserv.com
10	prifhjstv.mooo.com
12	ghcxncadnj.dyndns.org
14	vvgmudq.yi.org
16	editpjxmgdy.dynserv.com
18	ruxujk.mooo.com
1a	bdaxel.dyndns.org
1c	jitymoptq.yi.org
1e	tjgpkvutob.dynserv.com
20	agdwgsptbxo.mooo.com
22	hvjvigrwd.dyndns.org
24	laebdbptt.yi.org

[Wireshark] Example: Kraken Network Traffic Decryption

Kraken cryptography model has a weakness

- » Keys derived from IP address, CPU details, HDD serial
- » Dummies! The keys are in the packet header
- » A Wireshark plugin can decrypt data in live or still pcaps

```
typedef struct MSG_HDR {
    unsigned char type[2];
    unsigned short ver;
    int length;
} MSG_HDR;

typedef struct PACKET {
    unsigned int key0;
    unsigned int key4;
    unsigned int key8;
    MSG_HDR msg;
    unsigned int chksum;
    unsigned char data[1];
} PACKET;
```

Key Criteria	Generation Methodology
IP addresses	API: GetAdaptersInfo
CPU Vendor ID	cpuid (eax = 0)
CPU Info and Feature Bits	cpuid (eax = 1)
CPU Serial Number	cpuid (eax = 3)
C:\ Volume Serial Number	API: GetVolumeInformation

[Wireshark] Example: Kraken Network Traffic Decryption

85 74.482548 66.29.87.159 192.168.2.5 UDP Source port: ddm-dfm Destination port: remote-as

- Frame 85 (298 bytes on wire, 298 bytes captured)
- Ethernet II, Src: AsustekC_61:5b:f1 (00:e0:18:61:5b:f1), Dst: 00:54:00:12:34
- Internet Protocol, Src: 66.29.87.159 (66.29.87.159), Dst: 192.168.2.5 (192.168.2.5)
- User Datagram Protocol, Src Port: ddm-dfm (447), Dst Port: remote-as (1053)
- Kraken Protocol**
 - key0: 0x8e8d6e1c
 - key4: 0x34ede158
 - key8: 0x723e3212
 - Type 0: 0x14 ("Module catalog")**
 - Type 1: 0x00
 - Version: 315
 - Payload Length: 232

Brought to you by MHL and Greg from iDefense. Enjoy.

0000	8e 8d 6e 1c 34 ed e1 58 72 3e 32 12 14 00 3b 01	..n.4..X r>2...;.
0010	e8 00 00 00 ec 20 b1 b9 3c 6d 6f 64 75 6c 65 73 <modules
0020	3e 3c 6d 6f 64 75 6c 65 3e 3c 76 65 72 73 69 6f	><module ><versio
0030	6e 3e 31 31 33 3c 2f 76 65 72 73 69 6f 6e 3e 3c	n>113</v ersion><
0040	6e 61 6d 65 3e 69 6d 6d 6f 64 75 6c 65 3c 2f 6e	name>imm odule</n
0050	61 6d 65 3e 3c 6d 6f 64 75 6c 65 69 64 3e 31 31	ame><mod uleid>11
0060	32 3c 2f 6d 6f 64 75 6c 65 69 64 3e 3c 63 72 63	2</modul eid><crc
0070	3e 33 32 35 30 38 36 34 39 35 33 3c 2f 63 72 63	>3250864 953</crc
0080	3e 3c 2f 6d 6f 64 75 6c 65 3e 3c 6d 6f 64 75 6c	></modul e><modul
0090	65 3e 3c 76 65 72 73 69 6f 6e 3e 31 30 33 3c 2f	e><versi on>103</
00a0	76 65 72 73 69 6f 6e 3e 3c 6e 61 6d 65 3e 73 6e	version> <name>sn
00b0	69 66 66 6d 6f 64 75 6c 65 3c 2f 6e 61 6d 65 3e	iffmodul e</name>
00c0	3c 6d 6f 64 75 6c 65 69 64 3e 31 31 35 3c 2f 6d	<modulei d>115</m

Frame (298 bytes) | Decoded Payload (Client Mode) (232 bytes) | Decoded Payload (Server Mode) (232 bytes)

[Passive] Example: Laqma IPC Snooping

Two usermode components: client (DLL) and server (EXE)

- » **Client DLL hooks API functions**
 - » PFXImportCertStore (certificates)
 - » HttpSendRequest (login credentials)
 - » Writes data to “Global\temp_xchg”
 - » Sends “data waiting” message to __axelsrv
- » **Server EXE monitors messages to __axelsrv**
 - » Maps “Global\temp_xchg” to acquire data
 - » Dispatches attack based on data type

[Passive] Example: Laqma IPC Snooping

```
BOOL HOOK_HttpSendRequest(HINTERNET hRequest,
                          LPCTSTR lpszHeaders, DWORD dwHeadersLength,
                          LPVOID lpOptional, DWORD dwOptionalLength)
{
    PTEMP_XCHG  lpvData;
    HWND        hwnd;
    BOOL        bStatus;

    //Call the legitimate API function
    bStatus = HttpSendRequest(hRequest,
                              lpszHeaders,
                              dwHeadersLength,
                              lpOptional,
                              dwOptionalLength);

    //Write the stolen data to shared memory segment
    lpvData = (PTEMP_XCHG) MapSharedMemory("Global\\temp_xchg");
    if (lpvData != NULL && lpOptional != NULL) {
        lpvData->uMsg = HTTPSENDREQUESTAW;
        memcpy(&lpvData->Buff.Handle, &hRequest, sizeof(HANDLE));
        memcpy(&lpvData->Buff.Data, lpOptional, strlen(lpOptional));
        //Send the message to server that the API was triggered
        hwnd = FindWindow(TEXT("__intsrv32"), NULL);
        SendMessage(hwnd, HTTPSENDREQUESTAW, 0, 0);
    }

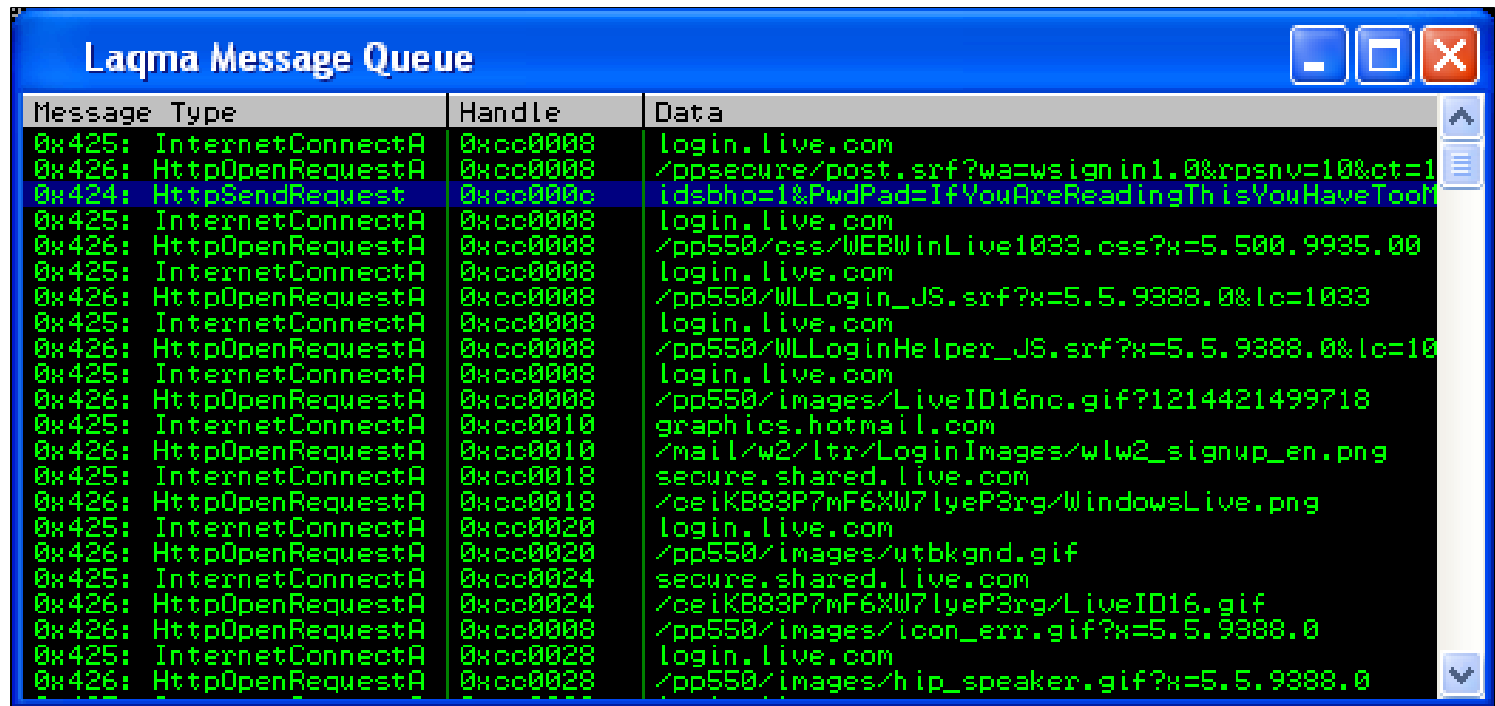
    return bStatus;
}
```

[Passive] Example: Laqma IPC Snooping

Script fundamentals

- » Laqma calls MapViewOfFile to locate “Global\temp_xchg” and then writes data to the region
- » If we hook MapViewOfFile, it will occur before the write
- » Also don't want to hook internal write function
 - » Laqma uses internal memcpy-like write function
 - » The write function address may change
- » We can hook FlushViewOfFile instead
 - » First parameter is a pointer to memory region
 - » Parse the values for validity before interpreting

DEMO



Message Type	Handle	Data
0x425: InternetConnectA	0xcc0008	login.live.com
0x426: HttpOpenRequestA	0xcc0008	/ppsecure/post.srf?wa=signin1.0&rpsnv=10&ct=1
0x424: HttpSendRequest	0xcc000c	idsbho=1&PwdPad=IfYouAreReadingThisYouHaveTooM
0x425: InternetConnectA	0xcc0008	login.live.com
0x426: HttpOpenRequestA	0xcc0008	/pp550/css/WEBWinLive1033.css?x=5.500.9935.00
0x425: InternetConnectA	0xcc0008	login.live.com
0x426: HttpOpenRequestA	0xcc0008	/pp550/MLLogin_JS.srf?x=5.5.9388.0&lc=1033
0x425: InternetConnectA	0xcc0008	login.live.com
0x426: HttpOpenRequestA	0xcc0008	/pp550/MLLoginHelper_JS.srf?x=5.5.9388.0&lc=10
0x425: InternetConnectA	0xcc0008	login.live.com
0x426: HttpOpenRequestA	0xcc0008	/pp550/images/LiveID16nc.gif?1214421499718
0x425: InternetConnectA	0xcc0010	graphics.hotmail.com
0x426: HttpOpenRequestA	0xcc0010	/mail/w2/ltr/LoginImages/wlw2_signup_en.png
0x425: InternetConnectA	0xcc0018	secure.shared.live.com
0x426: HttpOpenRequestA	0xcc0018	/ceiKB83P7mF6XW7lyeP3rg/WindowsLive.png
0x425: InternetConnectA	0xcc0020	login.live.com
0x426: HttpOpenRequestA	0xcc0020	/pp550/images/utbknd.gif
0x425: InternetConnectA	0xcc0024	secure.shared.live.com
0x426: HttpOpenRequestA	0xcc0024	/ceiKB83P7mF6XW7lyeP3rg/LiveID16.gif
0x426: HttpOpenRequestA	0xcc0008	/pp550/images/icon_err.gif?x=5.5.9388.0
0x425: InternetConnectA	0xcc0028	login.live.com
0x426: HttpOpenRequestA	0xcc0028	/pp550/images/hip_speaker.gif?x=5.5.9388.0

[Passive] Example: Torpig/MBR Configuration Decryption

- **Torpig incorporates MBR rootkit**
 - » Based on eEye BootRoot
- **Kernel driver downloads encrypted files over HTTP**
 - » DLL that hooks API functions
 - » Configuration file containing targeted URLs, drop sites, etc.
- **Decryption occurs in kernel space**
 - » The configuration is available to the DLL via `\\.\pipe!\win$`
 - » But any usermode process can query the named pipe

Script fundamentals

- » **Pipe interactions occur when DLL loads**
 - » Attach to running process will not work (too late)
- » **We can hook CreateFile and wait for \\.\pipe\!win\$**
 - » Then activate ReadFile and WriteFile hooks
 - » Verify that the handle belongs to the right object
 - » Perform analysis on data sent/received

```
if function_name == "CreateFileW":
    addr = imm.readLong(regs['ESP'] + 0x4)
    filename = imm.readWString(addr)
    filename = filename.replace("\x00", "")
    if filename == "\\.\pipe\!win$":
        self.add("readfile", imm.getAddress("kernel32.ReadFile"))
        self.add("writefile", imm.getAddress("kernel32.WriteFile"))
```

DEMO

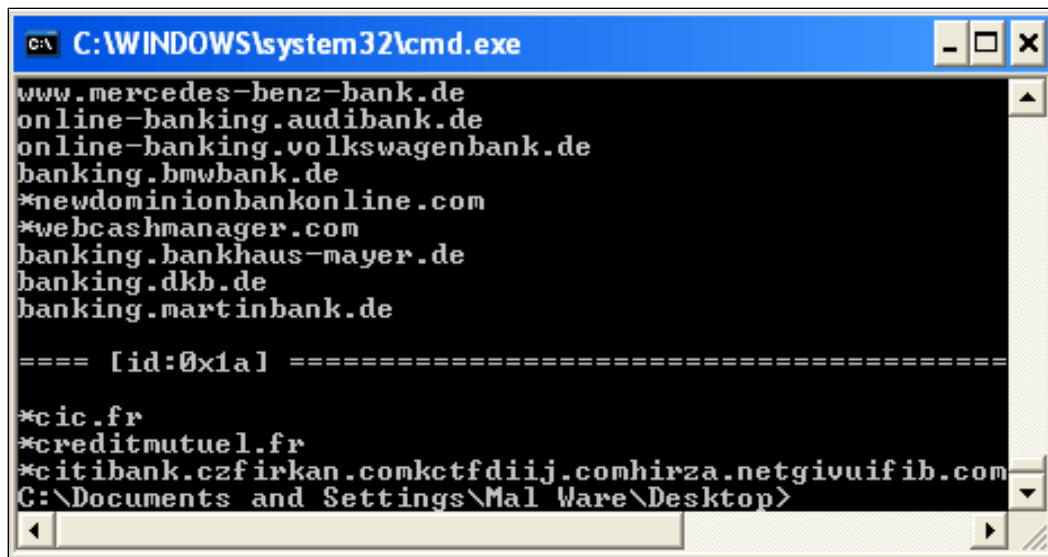
Torpig Spy			
Hook	Size	Data	Description
Read	0x1	N/A	< Request Ack
Read	0x4	N/A	< Response size
Read	0x9	N/A	< Response
Write	0x1	01	> Client Hello 1
Write	0x1	02	> Client Hello 2
Write	0x4	06 00 00 00	> Request size (6 bytes)
Write	0x6	00 00 00 00 11 00	> Request structure (Type 17)
Read	0x1	N/A	< Request Ack
Read	0x4	N/A	< Response size
Read	0x9	N/A	< Response
Write	0x1	01	> Client Hello 1
Write	0x1	02	> Client Hello 2
Write	0x4	06 00 00 00	> Request size (6 bytes)
Write	0x6	00 00 00 00 12 00	> Request structure (Type 18)
Read	0x1	N/A	< Request Ack
Read	0x4	N/A	< Response size
Read	0x2112	N/A	< Response

[Passive] Example: Torpig/MBR Configuration Decryption

- A stand-alone program can read the data
 - » Use the protocol displayed in script output
 - » Increment the reqType for each iteration

```
#define DOMAIN_TARGETS_A 0x17
#define DOMAIN_TARGETS_B 0x18
#define DOMAIN_TARGETS_C 0x19
#define DOMAIN_TARGETS_D 0x1a

typedef struct pipeCONTROL {
    BYTE    helloStatus;
    BYTE    Reserved0[3];
    BYTE    reqType;
    BYTE    Reserved1;
} PIPE_CONTROL;
```



```
C:\WINDOWS\system32\cmd.exe
www.mercedes-benz-bank.de
online-banking.audibank.de
online-banking.volkswagenbank.de
banking.bmwbank.de
*newdominionbankonline.com
*webcashmanager.com
banking.bankhaus-mayer.de
banking.dkb.de
banking.martinbank.de

==== [id:0x1a] =====
*cic.fr
*creditmutuel.fr
*citibank.czfirkan.comkctfdij.comhirza.netgivuifib.com
C:\Documents and Settings\Mal Ware\Desktop>
```

Reconnaissance and Utility Scripting

- **Facilitate RCE with useful scripts**
 - » Importing and exporting data (IDA <-> Immdbg)
 - » Scanning arbitrary files for shellcode
 - » Scanning memory for API function hooks

[Utility] Importing and Exporting Data

- **Transfer data into IDA for static analysis**
 - » Save the debugger script's output to text file
 - » Use IDAPython to import the data to the .idb
 - » Patch decoded strings, rename call tables with API
- **Transfer data into Immdbg for dynamic analysis**
 - » Reverse the process above
 - » Save named functions, comments, structure members

DEMO

```
100242D8 . E8 33F6FFFF CALL p.10023910
100242DD . 8B85 70FFFFFF MOV EAX,DWORD PTR SS:[EBP-90]
100242E3 . E8 4C05FEFF CALL p.10004834
100242E8 . 50 PUSH EAX
100242E9 . A1 B8A00210 MOV EAX,DWORD PTR DS:[1002AAB8]
100242EE . 50 PUSH EAX
100242EF . E8 7025FEFF CALL <JMP.&kernel32.GetProcAddress>
100242F4 . A3 34A00210 MOV DWORD PTR DS:[1002AA34],EAX
100242F9 . 8D8D 6CFFFFFF LEA ECX,DWORD PTR SS:[EBP-94]
100242FF . 66:BA BC7C MOV DX,7CBC
10024303 . B8 CC460210 MOV EAX,p.100246CC
10024308 . E8 03F6FFFF CALL p.10023910
1002430D . 8B85 6CFFFFFF MOV EAX,DWORD PTR SS:[EBP-94]
10024313 . E8 1C05FEFF CALL p.10004834
10024318 . 50 PUSH EAX
10024319 . E8 B625FEFF CALL <JMP.&kernel32.LoadLibraryA>
1002431E . A3 C0A00210 MOV DWORD PTR DS:[1002AAC0],EAX
10024323 . 8D8D 68FFFFFF LEA ECX,DWORD PTR SS:[EBP-98]
10024329 . 66:BA BC7C MOV DX,7CBC
```

ProcNameOrOrdinal
hModule => NULL
GetProcAddress

ASCII "Kgq54jNDN7igM1C"

FileName
LoadLibraryA

```
100242D8 . E8 33F6FFFF CALL p.10023910
100242DD . 8B85 70FFFFFF MOV EAX,DWORD PTR SS:[EBP-90]
100242E3 . E8 4C05FEFF CALL <p.@System@@LStrToPChar$qqrX17Syst
100242E8 . 50 PUSH EAX
100242E9 . A1 B8A00210 MOV EAX,DWORD PTR DS:[1002AAB8]
100242EE . 50 PUSH EAX
100242EF . E8 7025FEFF CALL <p.GetProcAddress_00>
100242F4 . A3 34A00210 MOV DWORD PTR DS:[1002AA34],EAX
100242F9 . 8D8D 6CFFFFFF LEA ECX,DWORD PTR SS:[EBP-94]
100242FF . 66:BA BC7C MOV DX,7CBC
10024303 . B8 CC460210 MOV EAX,p.100246CC
10024308 . E8 03F6FFFF CALL p.10023910
1002430D . 8B85 6CFFFFFF MOV EAX,DWORD PTR SS:[EBP-94]
10024313 . E8 1C05FEFF CALL <p.@System@@LStrToPChar$qqrX17Syst
10024318 . 50 PUSH EAX
10024319 . E8 B625FEFF CALL <p.LoadLibraryA>
1002431E . A3 C0A00210 MOV DWORD PTR DS:[1002AAC0],EAX
10024323 . 8D8D 68FFFFFF LEA ECX,DWORD PTR SS:[EBP-98]
10024329 . 66:BA BC7C MOV DX,7CBC
```

ProcNameOrOrdinal
hModule => NULL
GetProcAddress

ASCII "Kgq54jNDN7igM1C"

FileName
LoadLibraryA

[Utility] Detect Shellcode in Arbitrary Files

- **Is that .pdf, .doc or .jpeg malicious?**
 - » Applies to any file type
 - » Could still be malicious without shellcode
 - » JavaScript bytecode
 - » PDF FlateDecode
- **Based on common characteristics of shellcode**
 - » Load the file into memory and scan each byte
 - » Is it a jump or a call instruction?
 - » Is the destination address valid?
 - » Are the instructions at the destination address valid?

[Utility] Detect Shellcode in Arbitrary Files

The image shows two windows from a debugger. The top window, titled "Shell Code Detect", displays a list of instructions with columns for "Rel/Abs Addr", "Op", "Dest Addr", and "Next Op". The instruction at address 0x180e91 is highlighted in blue. The bottom window, titled "CPU - main thread", shows the current instruction stream. The instruction at address 00180E91 is highlighted in blue and matches the instruction in the Shell Code Detect window. A red arrow points from this instruction in the CPU window to the corresponding entry in the Shell Code Detect window.

Rel/Abs Addr	Op	Dest Addr	Next Op
0x2c1d8 (0x17c1d8)	JMP SHORT 0017C21F	0x17c21f	ADC ECX,DWORD PTR DS:[ESI] ; MOV EBP,DWORD PTR DS:[EBP]
0x2c728 (0x17c728)	JMP SHORT 0017C77F	0x17c77f	LODS DWORD PTR DS:[ESI] ; LODS DWORD PTR DS:[ESI]
0x2d587 (0x17d587)	JMP SHORT 0017D5E9	0x17d5e9	NOT DWORD PTR DS:[ESI] ; ADD DH,CH
0x2eb69 (0x17eb69)	JMP SHORT 0017EBC3	0x17ebc3	POP EBX ; OUT DX,AL
0x2f390 (0x17f390)	JMP SHORT 0017F406	0x17f406	INTO : XCHG EAX,ECX
0x2f3e6 (0x17f3e6)	JMP SHORT 0017F42A	0x17f42a	CALL E3193A9F ; CMPS DWORD PTR DS:[ESI],DWORD PTR DS:[EDI]
0x30e53 (0x180e53)	CALL 00180E58	0x180e58	POP EBX ; ADD EBX,FFBFEF54
0x30e5f (0x180e5f)	JMP SHORT 00180E91	0x180e91	MOV EAX,DWORD PTR FS:[30] ; MOV EAX,DWORD PTR DS:[EAX]
0x30eb2 (0x180eb2)	CALL 00180FA7	0x180fa7	PUSHAD ; MOV ESI,DWORD PTR DS:[EBX+4010D]
0x30ed7 (0x180ed7)	JMP SHORT 00180EF3	0x180ef3	CMP ESI,2000 ; JB SHORT 00180ED9
0x30ef0 (0x180ef0)	JMP SHORT 00180F00	0x180f00	MOV DWORD PTR DS:[EBX+4010D9],ESI ; PUSH DWORD PTR DS:[EBX+4010D9]
0x30efb (0x180efb)	JMP 00180F92	0x180f92	PUSH DWORD PTR DS:[EBX+4010D9] ; CALL DWORD PTR DS:[EBX+4010D9]
0x30f1d (0x180f1d)	JMP SHORT 00180F92	0x180f92	PUSH DWORD PTR DS:[EBX+4010D9] ; CALL DWORD PTR DS:[EBX+4010D9]
0x30f1f (0x180f1f)	JMP SHORT 00180F34	0x180f34	PUSH 0 ; PUSH 0
0x30f7c (0x180f7c)	JMP SHORT 00180F92	0x180f92	PUSH DWORD PTR DS:[EBX+4010D9] ; CALL DWORD PTR DS:[EBX+4010D9]
0x30ff1 (0x180ff1)	JMP SHORT 0018100C	0x18100c	SUB EDX,DWORD PTR DS:[ESI+20] ; SUB EDX,DWORD PTR DS:[ESI+20]
0x31039 (0x181039)	JMP SHORT 0018103C	0x18103c	POPAD ; RETN
0x32012 (0x182012)	JMP SHORT 00182019	0x182019	NOP ; PREFIX REPNE:

Address	Op
00180E91	MOV EAX,DWORD PTR FS:[30]
00180E97	MOV EAX,DWORD PTR DS:[EAX+C]
00180E9A	MOV ESI,DWORD PTR DS:[EAX+1C]
00180E9D	LODS DWORD PTR DS:[ESI]
00180E9E	MOV EAX,DWORD PTR DS:[EAX+8]
00180EA1	MOV DWORD PTR DS:[EBX+4010D1],EAX
00180EA7	CLD
00180EA8	LEA EDI,DWORD PTR DS:[EBX+4010B5]
00180EAE	XOR ECX,ECX
00180EB0	MOV CL,7
00180EB2	CALL 00180FA7

[Utility] Detect Shellcode in Arbitrary Files

- Result set is proportional to file size
- Other methods/resources for detecting shellcode
 - » Open suspect file in IDA and press “c”
 - » Use a stream disassembler
 - » diStorm64 (<http://www.ragestorm.net/distorm>)
 - » Polymorphic Shellcode (Detection) by Christoph Gratl
 - » Hybrid Engine for Polymorphic Shellcode Detection by Payer, Teufl and Lamberger
 - » Libemu (<http://libemu.mwcollect.org>)

[Utility] Detect Hooked API Functions

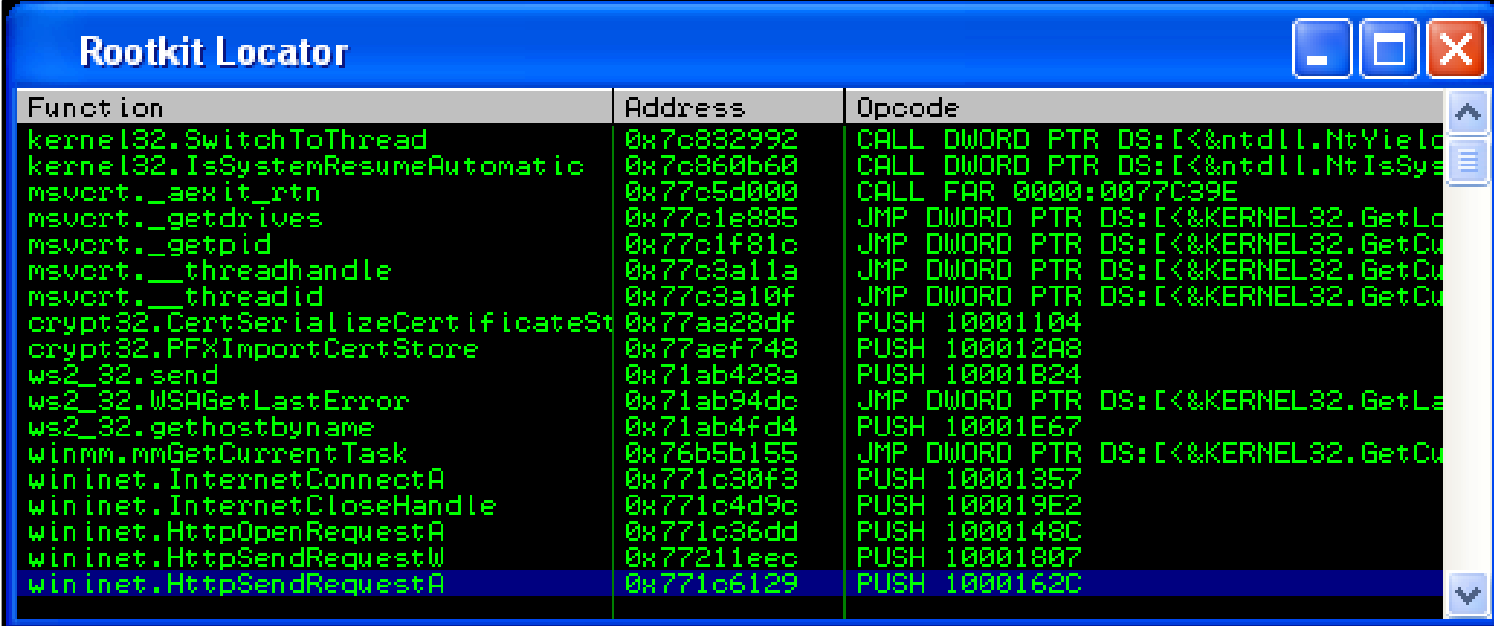
- **Multiple methods for user mode API hooking**
 - » **Trampoline hooks**
 - » **IAT hooks (watch out for LoadLibrary/GetProcAddress hooks)**
 - » **Modify DLL on disk**
- **It is easy to check for trampoline hooks**
 - » **For each loaded DLL**
 - » **For each exported function**
 - » **Is the prologue overwritten?**
 - » CALL
 - » JMP
 - » PUSH/RET
 - » **Yes – is the destination inside the loaded DLL?**

[Utility] Detect Hooked API Functions

- Determine the purpose of an API hook
 - » Set breakpoints on the hooked functions
 - » Use the target process as desired (i.e., browse to Web page)
 - » Debug

```
addr = imm.getAddress(string)
if addr != -1:
    op = imm.disasm(addr)
    instr = op.getDisasm()
    # Check for hooks of type "jmp Oxdeadbeef" or "call Oxdeadbeef"
    if op.isJump() or op.isCall():
        dest = op.getJumpAddr()
        if isExternalToModule(imm, addr, dest) == True:
            table.add('', ['%s' % string, '0x%x' % addr, '%s' % instr])
    # Check for hooks of type "push Oxdeadbeef; retn"
    elif op.isPush():
        nextop = imm.disasm(addr + op.getSize())
        if nextop.isRet():
            call_dest = imm.readLong(addr+op.getSize()+1)
            if isExternalToModule(imm, addr, call_dest):
                table.add('', ['%s' % string, '0x%x' % addr, '%s' % instr])
```

DEMO



The screenshot shows a window titled "Rootkit Locator" with a table of detected hooked API functions. The table has three columns: Function, Address, and Opcode. The functions listed include kernel32.SwitchToThread, kernel32.IsSystemResumeAutomatic, msvrt._aexit_rtn, msvrt._getdrives, msvrt._getpid, msvrt.__threadhandle, msvrt.__threadid, crypt32.CertSerializeCertificateSt, crypt32.PFXImportCertStore, ws2_32.send, ws2_32.WSAGetLastError, ws2_32.gethostbyname, winmm.mmGetCurrentTask, wininet.InternetConnectA, wininet.InternetCloseHandle, wininet.HttpOpenRequestA, wininet.HttpSendRequestW, and wininet.HttpSendRequestA. The last row is highlighted in blue.

Function	Address	Opcode
kernel32.SwitchToThread	0x7c832992	CALL DWORD PTR DS:[<&ntdll.NtYield
kernel32.IsSystemResumeAutomatic	0x7c860b60	CALL DWORD PTR DS:[<&ntdll.NtIsSys
msvrt._aexit_rtn	0x77c5d000	CALL FAR 0000:0077C39E
msvrt._getdrives	0x77c1e885	JMP DWORD PTR DS:[<&KERNEL32.GetLo
msvrt._getpid	0x77c1f81c	JMP DWORD PTR DS:[<&KERNEL32.GetCu
msvrt.__threadhandle	0x77c3a11a	JMP DWORD PTR DS:[<&KERNEL32.GetCu
msvrt.__threadid	0x77c3a10f	JMP DWORD PTR DS:[<&KERNEL32.GetCu
crypt32.CertSerializeCertificateSt	0x77aa28df	PUSH 10001104
crypt32.PFXImportCertStore	0x77aef748	PUSH 100012A8
ws2_32.send	0x71ab428a	PUSH 10001B24
ws2_32.WSAGetLastError	0x71ab94dc	JMP DWORD PTR DS:[<&KERNEL32.GetLa
ws2_32.gethostbyname	0x71ab4fd4	PUSH 10001E67
winmm.mmGetCurrentTask	0x76b5b155	JMP DWORD PTR DS:[<&KERNEL32.GetCu
wininet.InternetConnectA	0x771c30f3	PUSH 10001357
wininet.InternetCloseHandle	0x771c4d9c	PUSH 100019E2
wininet.HttpOpenRequestA	0x771c36dd	PUSH 1000148C
wininet.HttpSendRequestW	0x77211eec	PUSH 10001807
wininet.HttpSendRequestA	0x771c6129	PUSH 1000162C

Potential Caveats

- The project may require “manual” or “assisted” RCE
 - » Wireshark, Glamour/Gpcoder tool
- Scripting ring-zero malicious code is possible, but challenging
 - » Patch API calls (ntoskrnl.exe:RtlTimeToTimeFields -> ntdll.dll:RtlTimeToFileFields)
- Interpreted scripts can take a while to execute
- Subject to anti-debugger detection
 - » But try <http://www.PEiD.info/BobSoft>

Additional Resources

- Immunity Debugger and Forums
 - » <http://www.immunitysec.com/products-immdbg.shtml>
 - » <http://forum.immunityinc.com>
- OpenRCE has a few scripts, too
 - » <http://www.openrce.org>
- We're hosting ours with Google Code
 - » <http://code.google.com/p/mhl-malware-scripts>

Q and A

Michael Ligh

Greg Sinclair

iDefense Security Intelligence Services